Exact Complexity Results and Polynomial-Time Algorithms for Derivative Accumulation

Andrew Lyons

Computation Institute, University of Chicago, and
Mathematics and Computer Science Division, Argonne National Laboratory
lyonsam@gmail.com

Abstract

We discuss the complexity of evaluating certain collections of monic multilinear polynomials using operations in $\{\times, +\}$. The functions we consider, which are defined on paths in directed acyclic graphs (DAGs), represent derivative computations that are based on the chain rule and have direct applications in high-performance scientific computing. Our main results concern functions derived from single-source, single-sink DAGs whose maximal paths all have length three. We derive tight, exact lower bounds for the numbers of multiplications, additions, and total arithmetic operations needed. Moreover, we show that, given such a DAG, an arithmetic circuit (or straight-line program) of minimum size that evaluates $\mathcal{J}(G)$ can be constructed in polynomial time. In contrast, we show the (perhaps surprising) result that the problem of finding a circuit of minimum size for a given DAG becomes NP-hard, even for the restricted class of DAGs considered in this paper, when some subset of the arcs may be labeled with the multiplicative identity 1 rather than an indeterminate.

1 Introduction

Let G be a finite directed acyclic graph (DAG) with sources $s_1, \ldots, s_{\tilde{n}}$ and sinks $t_1, \ldots, t_{\tilde{m}}$. When the arcs $(u, v) \in A(G)$ are labeled with indeterminates x_{uv} from a set X, the *structural derivative* $\mathcal{J}(G)$ of G is a matrix defined for all $i \in \{1, \ldots, \tilde{n}\}, j \in \{1, \ldots, \tilde{m}\}$ as

$$\mathcal{J}_{ij}(G) = \sum_{P \in [s_i \rightsquigarrow t_j]} \prod_{(u,v) \in P} x_{uv},$$

where $[s_i \rightsquigarrow t_j]$ denotes the set of all paths from source s_i to sink t_j in G. We assume that the indeterminates labeling the arcs are algebraically independent and that no two edges are labeled with the same indeterminate. We are interested in the exact complexity of evaluating $\mathcal{J}(G)$ using operations in $\{\times, +\}$.

Our study of the structural derivative function is motivated by applications in the field of automatic (or algorithmic) differentiation (AD) [7], a technique for obtaining numerical derivatives that is used widely in computational science and engineering. In AD, the chain rule from calculus is used to obtain derivatives of a function $\tilde{F} \colon \mathbb{R}^{\tilde{n}} \to \mathbb{R}^{\tilde{m}}$ that is given in the form of an *evaluation procedure*, such as a straight-line program in an imperative programming language. From such a program, we construct the computational graph G, a DAG that describes the way in which the outputs (sinks) are computed from the inputs (sources) as compositions of arbitrary nonlinear functions. When the arcs (u, v) are labeled with indeterminates corresponding to the *local partial derivatives* $\frac{\partial v}{\partial u}$ of these functions, a classic result (apparently first due to Bauer [1]) states that the structural derivative function computes the Jacobian matrix of $\tilde{F}$. Evaluating $\mathcal{J}(G)$ then amounts to *accumulating* each global partial derivative $\mathcal{J}_{ij}(G) = \frac{\partial t_j}{\partial s_i}$ from the local partial derivatives.

In the context of AD, the use of cancellations is considered undesirable for numerical reasons, and we therefore forbid division and subtraction. From an algebraic perspective, then, we work in the semiring $\langle \mathbb{R}, \times, +, 0, 1 \rangle$, which does not have additive inverses. In Section 4, we formalize the computational model that has been implicitly used for accumulation procedures in the literature. We consider the computation of $\mathcal{J}(G)$ by an *accumulation circuit*, which is a certain type of monotone arithmetic circuit that does not use constants from the underlying field.

draft version hg:60ec37863c64:182 compiled February 16, 2010

Results. In Section 5, we present the first exact lower bound on the complexity of $\mathcal{J}(G)$ for a nontrivial class of graphs. Our bound relates the multiplicative complexity of $\mathcal{J}(G)$ to the vertex cover number of an associated undirected bipartite graph. A tight upper bound is given in Section 6 along with a polynomial-time algorithm that, given a graph G from this class, produces an optimal accumulation circuit. We also give complete characterizations of the additive and total complexity of $\mathcal{J}(G)$ for the graphs that we consider.

The fact that our algorithm runs in polynomial time is to be contrasted with the following fact. The general problem of determining the minimum number of arithmetic operations (or just multiplications) required to evaluate $\mathcal{J}(G)$ for a given graph G has been conjectured for decades to be NP-hard [6]. Our lower bound represents a step toward resolving the complexity of this circuit minimization problem.

In Section 7, we consider a slight generalization of this problem in which a subset of the arcs in G may be labeled with the multiplicative unit 1. We show that the problem then immediately becomes NP-hard, even for DAGs that have the same structure as those addressed in Sections 5 and 6.

2 Background and Motivation

Suppose we are given a program for a function $\tilde{F}\colon \tilde{\mathbf{x}} \mapsto \tilde{\mathbf{y}}$ and asked to produce an augmented program for $\tilde{F}$ that can compute the value of not only $\tilde{\mathbf{y}}$ but also some derivative object related to $\tilde{F}$ (such as the Jacobian matrix) at a particular argument $\tilde{\mathbf{x}}_0$. Specialized compilers that implement AD techniques—such as Tapenade [8] and OpenAD [25]—accomplish this task in the following way. The key is that, in AD, we leverage the chain rule to differentiate an *algorithm* for $\tilde{F}$, rather than $\tilde{F}$ itself.

Evaluation procedures and linearization. We assume $\tilde{F}\colon \mathbb{R}^{\tilde{n}} \mapsto \mathbb{R}^{\tilde{m}}$ is a vector function given in the form of an *evaluation procedure* such as a straight-line program in which temporary values are computed and assigned to variables. The values for the first $\tilde{n}$ variables are assumed to be taken from the input vector $\tilde{\mathbf{x}} \in \mathbb{R}^{\tilde{n}}$, and the values of the output vector $\tilde{\mathbf{y}} \in \mathbb{R}^{\tilde{m}}$ are assumed to be extracted from the last $\tilde{m}$ variables. Each program variable v_j is assigned the value of an arbitary nonlinear function φ_j, which takes some subset of the previously computed variables as arguments. These direct dependencies among the variables induce the DAG G, where an arc $(v_i, v_j) \in A(G)$ indicates that variable v_j depends directly on variable v_i indicating $v_j = \varphi_j(\ldots, v_i, \ldots)$. The functions φ_j can be either built-in intrinsic functions (such as sin or exp) or arbitrary nonlinear functions that may even be implemented in separate subroutines. We require only that an explicit *linearization* procedure can be created for evaluating the local partial derivatives $\frac{\partial \varphi_j}{\partial v_i}$. This may be produced either by an automatic process (such as $\frac{\partial \sin(v)}{\partial v} = \cos(v)$) or by a hand-coded subroutine. Note that both the variables v_i and the local partials $\frac{\partial v_j}{\partial v_i}$ then take values in $\mathbb{R}$ when $\tilde{F}$ is evaluated at a particular argument $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{\tilde{n}}$. Linearization is a fundamental part of any AD compiler; we therefore assume that the linearization process has already been performed and that the arcs $A(G)$ have been labeled with indeterminates corresponding to the local partial derivatives. For our purposes, the values at the vertices in G are irrelevant and will be ignored.

Derivative accumulation. What remains is to apply the chain rule, which entails creating an *accumulation procedure* that will be appended to the evaluation and linearization procedures. An accumulation procedure describes how the chain rule will be applied to evaluate $\mathcal{J}(G)$, and may be thought of either as a straight-line program or as an arithmetic circuit. In either case, $\mathcal{J}(G)$ is considered to be computed over $\langle \mathbb{R}, \times, +, 0, 1 \rangle$.

Definition 1 (accumulation complexity)**.** *Let G be a DAG such that the arcs $A(G)$ are labeled with indeterminants from a set X. The* accumulation complexity $\mathbf{ACC}(G)$ *of G is the minimum number of arithmetic operations required to compute all the entries of $\mathcal{J}(G)$ over X using only operations in $\{\times, +\}$. The* multiplicative *accumulation complexity* $\mathbf{ACC}_\times(G)$ *and the* additive *accumulation complexity* $\mathbf{ACC}_+(G)$ *are the minimum numbers of multiplications and additions required, respectively.*

Multiplications are generally considered to be significantly more expensive than additions; hence, $\mathbf{ACC}_\times(G)$ is the primary complexity measure of interest. Note that the number of terms in any of the polynomials that $\mathcal{J}(G)$ comprises could be exponential in the number of arcs in G. Nevertheless, $\mathcal{J}(G)$ can always

be evaluated at a cost of only polynomially many multiplications and additions. The *forward* and *reverse* modes of AD are often used in practice. The *forward mode* yields rows of $\mathcal{J}(G)$ at a cost of roughly $|A(G)|$ multiplications each (this cost is indepenent of $\tilde{m}$), while the *reverse mode*, which works symmetrically, yields columns of $\mathcal{J}(G)$ at a cost of roughly $|A(G)|$ multiplications each (independent of $\tilde{n}$). Combining these two approaches, we get $\mathbf{ACC}_{\times}(G) \leq |A(G)| * \min(\tilde{n}, \tilde{m})$.

Motivation. Large-scale numerical simulations are often written in imperative programming laguages such as C or Fortran. Such programs usually involve control flow structures such as branches, loops, and subroutine calls. In this context, $\tilde{F}$ typically represents not the entire function of interest, but rather a relatively small small block of code that may be contained in a subroutine that is called within a deeply nested loop structure. As a result, the code for $\tilde{F}$ (as well as the accumulation code) will be evaluated many times at different arguments for a single run of the larger program. In this way, saving even a small number of multiplication operations in the accumulation process has the potential to significantly reduce the overall runtime of the resulting program.

3 k-Homogeneous DAGs and Matrix Multiplication

Definition 2 (k-homogeneous DAG)**.** *A DAG is* k-homogeneous *if every path from a source to a sink has length exactly* k*. (In other words, each maximal path consists of exactly* k *edges.)*

When G is a k-homogeneous DAG, the structural derivative of G corresponds to a chained product of k sparse matrices in the following way, where a matrix is *sparse* if it has entries from $X \cup \{0\}$. This correspondence has also been observed by Vassilevska [26] and earlier by Mahr [13], Yuster and Zwick [27], and others. Let $V(G)$ be partitioned into $V^0, \ldots, V^k$ such that the vertices in every $V^\ell = \{v_1^\ell, \ldots, v_{n_\ell}^\ell\}$ are at distance ℓ from the sources. Every arc then has the form $(v_i^{\ell-1}, v_j^\ell)$; we denote the associated indeterminate by x_{ij}^ℓ. We then have $\mathcal{J}(G) = X^1 X^2 \cdots X^k$, where each X^ℓ is a $(n_{\ell-1} \times n_\ell)$ matrix whose entry in the ith row and jth column is x_{ij}^ℓ if $(v_i^{\ell-1}, v_j^\ell) \in A(G)$ and 0 otherwise. Each vertex v_i^ℓ then corresponds to the ith column of matrix X^ℓ and the ith row of matrix $X^{\ell+1}$.We will refer interchangeably to X^ℓ both as a subset of X and as a matrix; The meaning will be obvious from the context. Let $|X^\ell|$ denote the number of elements in the set X^ℓ as well as the number of nonzero entries in the matrix X^ℓ. For all $\ell \in \{1, \ldots, k\}$, let G^ℓ denote the *undirected* bipartite graph defined by

$$V\left(G^\ell\right) = V^{\ell-1} \cup V^\ell,$$
$$E\left(G^\ell\right) = \left\{\{v_i^{\ell-1}, v_j^\ell\} \mid (v_i^{\ell-1}, v_j^\ell) \in A(G)\right\}.$$

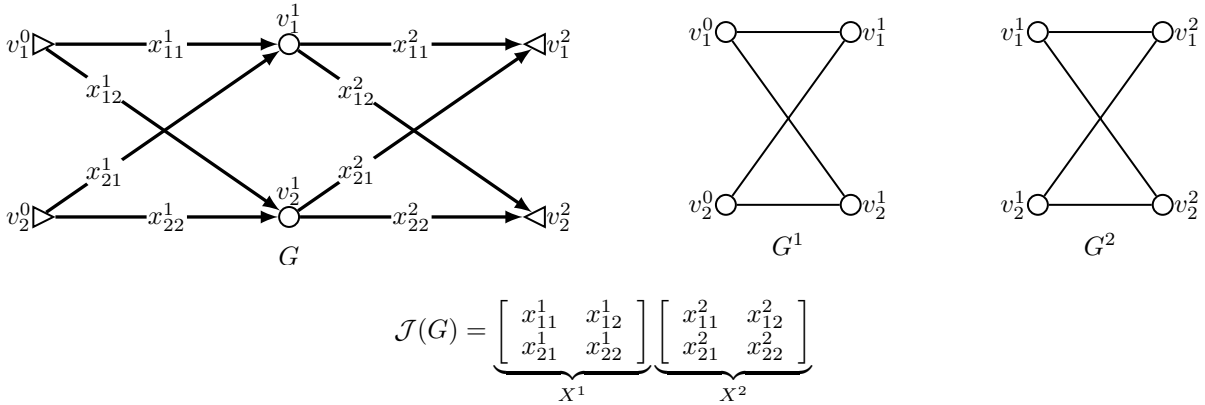


Figure 1: A 2-homogeneous DAG where $\mathcal{J}(G) = X^1 X^2$.

3.1 Tight lower bounds for matrix multiplication

There are a few known results on the monotone complexity of chained matrix products; all such results suggest that the schoolbook method for matrix multiplication is optimal (and unique up to isomorphism).

Theorem 1 ([10]). *$(k-1)n^3$ multiplications are necessary and sufficient to evaluate the product $A^1A^2\cdots A^k$ of k dense $n\times n$ matrices over $\langle\mathbb{R},\times,+,0,1\rangle$.*

For $k=2$, Theorem 1 is implied by the following stronger result.

Theorem 2 ([21, 20, 15, 14]). *If A is an $n_0\times n_1$ matrix and B is an $n_1\times n_2$ matrix, then $n_0n_1n_2$ multiplications and $n_0(n_1-1)n_2$ additions are necessary and sufficient to evaluate AB over any semiring of characteristic zero.*

Corollary 3.

(i) *If G is a 2-homogeneous DAG such that both G^1 and G^2 are complete bipartite graphs, then $\mathbf{ACC}_\times(G)=n_0n_1n_2$ and $\mathbf{ACC}_+(G)=n_0(n_1-1)n_2$;*

(ii) *If G is a k-homogeneous DAG such that $n_0=\cdots=n_k=n$ and $G^1,\ldots,G^k$ are complete bipartite graphs, then $\mathbf{ACC}_\times(G)=(k-1)n^3$.*

3.2 k-homogenous st-DAGs

Definition 3 (st-DAG). *An st-DAG is a DAG with unique source s and unique sink t.*

In this paper, we focus on 3-homogeneous st-DAGs G, for which $\mathcal{J}(G)=X^1X^2X^3$, where X^1 is a dense n_1-dimensional row vector, X^3 is a dense n_2-dimensional column vector, and X^2 is a sparse $(n_1\times n_2)$-dimensional matrix. We show that *every* accumulation circuit for such a DAG G uses $\mathbf{ACC}_+(G)=|X^2|-1$ additions and that $\mathbf{ACC}_\times(G)$ is equal to $|X^2|+\tau\big(G^2\big)$, where $\tau\big(G^2\big)$ is the vertex covering number of G^2. the undirected bipartite graph with adjacency matrix X^2. The forward mode is the same as performing the product as $\big(X^1X^2\big)X^3$, whereas the reverse mode performs $X^1\big(X^2X^3\big)$. Our result makes it easy to construct examples for which evaluating $\mathcal{J}(G)$ according to neither bracketing is optimal, thereby demonstrating the utility of exploiting *both* associativity and distributivity. On the other hand, we also demonstrate that every accumulation procedure for $\mathcal{J}(G)$ can be transformed into one that is noncommutative with no effect on the size; this implies that commutativity is powerless for 3-homogenous st-DAGs.

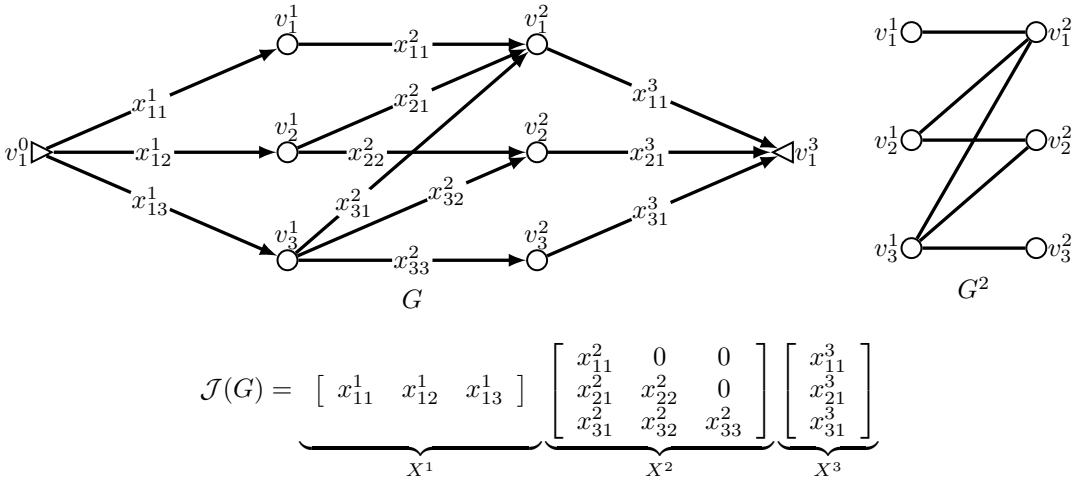


Figure 2: A 3-homogeneous st-DAG. $\mathcal{J}(G)=X^1X^2X^3$.

4 Computational Model

In this section, we define the computational model that we will use in proving our bounds. We work in the semiring $\langle \mathbb{R}, \times, +, 0, 1 \rangle$, which has the following properties:

- associativity of $\times$ and $+$;
- commutativity of $\times$ and $+$;
- distributivity of $\times$ over $+$;
- additive identity/multiplicative annihilator 0; and
- multiplicative identity 1.

In our case, the operations $\times$ and $+$ are just the usual multiplication and addition of real numbers. Note that we do not assume the existence of additive inverses. As a consequence, we assume that no cancellations occur in our computation. To avoid confusion, we use tree terminology (node, parent, child, ancestor, etc.) for circuits, reserving digraph terms (vertex, arc, inarc, etc.) for G itself.

4.1 Arithmetic circuits

Let $\mathbb{F}$ be a field and X be a set of indeterminates. An *arithmetic circuit Φ over X and $\mathbb{F}$* is a directed acyclic graph whose nodes are either *inputs* with zero children or *gates* with exactly two children The inputs take labels from $X \cup \mathbb{F}$. Every gate is labeled with an operation in $\{\times, +\}$ and computes a polynomial in the ring $\mathbb{F}[X]$ in the natural way: gates labeled $\times$ compute the product of their children; gates labeled $+$ compute the sum of their children. The *size* of Φ, denoted $|\Phi|$, is the number of gates; $|\Phi|_\times$ and $|\Phi|_+$ denote the numbers of product and sum gates, respectively. For a node $\alpha \in \Phi$, X_α denotes the set of inputs in Φ that have α as an ancestor, and $\mathrm{mon}(\alpha)$ denotes the set of monomials of the polynomial computed at α.

Definition 4 (monotone polynomial/circuit)**.** *A polynomial is* monotone *if the coefficient of every monomial is non-negative. An arithmetic circuit is* monotone *if every field element that occurs as a label on an input is positive.*

Monotone circuits can compute only monotone functions and are exactly those arithmetic circuits that don't use any negative constants from the underlying field. The following observation captures the intuition that nothing is destroyed in monotone computations.

Observation 4.1. *If α and ζ are nodes in a monotone arithmetic circuit such that γ is an ancestor of α, then for every $A \in \mathrm{mon}(\alpha)$ there exists some $Z \in \mathrm{mon}(\zeta)$ such that $A \subseteq Z$.*

Definition 5 (multilinear polynomial/circuit [19])**.** *A polynomial is* multilinear *if, in every monomial, the degree of every indeterminate is 1. An arithmetic circuit is* multilinear *if the polynomial computed at every node is multilinear.*

The following property is also known as *syntactic multilinearity* [22].

Definition 6 (multiplicatively disjoint [11])**.** *An arithmetic circuit is* multiplicatively disjoint *if $X_\alpha \cap X_\beta = \emptyset$ for every product gate $\rho = \alpha \times \beta$.*

Observation 4.2 ([19])**.** *Every monotone arithmetic circuit for a multilinear polynomial is multiplicatively disjoint.*

Definition 7 (monic polynomial/circuit)**.** *A polynomial is* monic *if the coefficient in every monomial is either 0 or 1. A* monic *arithmetic circuit computes a monic polynomial at every node.*

Every monic arithmetic circuit is also monotone. Since our operations are commutative and a monic multilinear polynomial is uniquely determined by the set of sets representation of its monomials, we don't distinguish between a monomial and the set consisting of those indeterminates that occur in it.

Definition 8 (additively disjoint)**.** *An arithmetic circuit Φ is* additively disjoint *if $\mathrm{mon}(\alpha) \cap \mathrm{mon}(\beta) = \emptyset$ for every sum gate $\sigma = \alpha + \beta$.*

Observation 4.3. *Every monic arithmetic circuit is additively disjoint.*

4.2 Accumulation circuits

According to the framework above, we can now give a formal description of the properties of accumulation circuits.

Definition 9 (accumulation circuit)**.** *An arithmetic circuit is an* accumulation circuit *if every input is labeled with an indeterminate (and not a constant from the underlying field).*

Like arithmetic circuits, each gate in an accumulation circuit computes a polynomial from the ring $\mathbb{F}[X]$.

Observation 4.4. *Every accumulation circuit for a collection of monic polynomials is monic.*

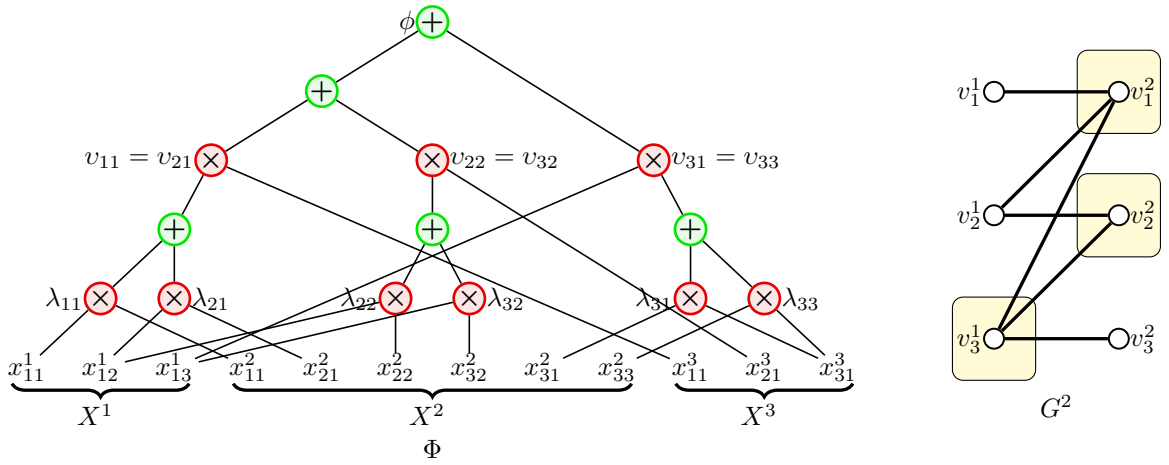


Figure 3: An accumulation circuit for the DAG G from the example shown in Figure 2. G^2 is shown with vertex cover $H = \{v^2_2, v^1_3, v^2_1\}$ indicated by squares.

Any accumulation circuit for $\mathcal{J}(G)$ must therefore be both monic and multilinear. Since we allow commutativity, each monomial is considered as a set of indeterminates. We now prove some elementary results concerning accumulation circuits that will be useful later. Let Φ be an accumulation circuit that computes a collection of monic multilinear polynomials.

Definition 10 (ζ-unique)**.** *A monomial $A \subseteq X$ is* ζ-unique *for a node $\zeta \in \Phi$ if there is exactly one monomial $Z \in \mathrm{mon}(\zeta)$ such that $A \subseteq Z$.*

Proposition 4.5. *Let α, ζ be nodes in Φ such that α is a descendant of ζ. If there is some $A \in \mathrm{mon}(\alpha)$ such that A is ζ-unique, then there is a unique path in Φ from α to ζ.*

Proof. Since Φ is multiplicatively disjoint, two distinct paths in Φ from α to ζ cannot meet at a product gate. Suppose two such paths meet at a sum gate σ. It follows that there exist distinct monomials $S_1, S_2 \in \mathrm{mon}(\sigma)$ such that $A \subseteq S_1$ and $A \subseteq S_2$. Since σ must be a descendant of ζ, Observation 4.1 implies that there are distinct $Z_1, Z_2 \in \mathrm{mon}(\zeta)$ such that $A \subseteq S_1 \subseteq Z_1$ and $A \subseteq S_2 \subseteq Z_2$, a contradiction. □

Lemma 4.6. *Let $\rho \in \Phi$ be a product gate with children α and β. If there is some $A \in \mathrm{mon}(\alpha)$ such that A is ζ-unique, then $|\mathrm{mon}(\beta)| = 1$.*

Proof. If there exist distinct $B_1, B_2 \in \mathrm{mon}(\beta)$, then $A \cup B_1, A \cup B_2 \in \mathrm{mon}(\rho)$, which contradicts the fact that A is ζ-unique. □

5 The Lower Bound

Throughout this section, Φ will denote an accumulation circuit that computes $\mathcal{J}(G)$ for a 3-homogeneous *st*-DAG G. We partition $V(G) = V^0 \cup V^1 \cup V^2 \cup V^3$ and $A(G) = X^1 \cup X^2 \cup X^3$ as in Section 3.

5.1 Additive complexity

For a node $\alpha \in \Phi$, define $X^2_\alpha \equiv X_\alpha \cap X^2$.

Proposition 5.1. *If σ is a sum gate in Φ with children α and β, then $X^2_\alpha \neq \emptyset$ and $X^2_\beta \neq \emptyset$.*

Proof. Suppose the claim is false, and assume without loss of generality that $X^2_\alpha = \emptyset$. Since $\mathrm{mon}(\sigma) = \mathrm{mon}(\alpha) \cup \mathrm{mon}(\beta)$, it follows that there exists some $S \in \mathrm{mon}(\sigma)$ such that $S \cap X^2 = \emptyset$. Since every monomial in $\mathrm{mon}(\phi)$ has a nonempty intersection with X^2, there must exist a product gate ρ in Φ such that (1) ρ is an ancestor of σ; (2) ρ has a children η, ω such that η is an ancestor of σ and ω is not; and (3) $X^2_\omega \neq \emptyset$. Since every $x \in X^2_\omega$ is ω-unique, it follows from Lemma 4.6 that $|\mathrm{mon}(\eta)| = 1$, which contradicts the fact that σ is a descendant of η, as $|\mathrm{mon}(\sigma)| = 1$. □

Lemma 5.2. $\mathbf{ACC}_+(G) = |X^2| - 1$. *Moreover, $|\Phi|_+ = |X^2| - 1$ for every accumulation circuit Φ computing $\mathcal{J}(G)$.*

Proof. We construct from Φ a binary tree T^+ whose leaves correspond to the inputs in X^2 and whose internal nodes correspond to the sum gates in Φ. Consider the circuit $T^{(2)}$ obtained by removing all nodes $\alpha \in \Phi$ for which $X^2_\alpha = \emptyset$. Since every node in $T^{(2)}$ computes a monomial that is ϕ-unique, $T^{(2)}$ is a tree with leaf set X^2. By Proposition 5.1, every sum gate σ in Φ in included in $T^{(2)}$, as are both children of σ. Now let ρ be an arbitrary product gate in $T^{(2)}$. Since no monomial in $\mathrm{mon}(\phi)$ contains more than one input from X^2, it follows that at most one child α of ρ is contained in $T^{(2)}$, so ρ has exactly one parent and exactly one child in $T^{(2)}$. Let T^+ be the tree obtained from $T^{(2)}$ by repeatedly removing every product gate ρ, followed by connecting the child of ρ directly to the parent. T^+ is a (full) binary tree whose internal nodes are exactly the sum gates in Φ and whose leaves are exactly those inputs from X^2. Since every full binary tree with k leaves has exactly $k-1$ internal nodes, our proof is complete. □

5.2 Multiplicative complexity

We begin by defining *parse trees*, which chart the production of particular monomials in $\mathrm{mon}(\phi)$.

Definition 11 (parse tree [10])**.** *A subcircuit T of Φ is a* parse tree *of Φ if it satisfies the following conditions:*

(i) *T contains the (unique) output of Φ.*
(ii) *If T contains a sum gate σ, then T contains exactly one of the children of σ.*
(iii) *If T contains a product gate ρ, then T contains both of the children of ρ.*
(iv) *No proper subtree of T satisfies* (i)-(iii).

Let $\mathrm{PT}(\Phi)$ denote the set of all parse trees of Φ. Every monomial of $\mathrm{mon}(\phi)$ is of the form $x^1_{1i} x^2_{ij} x^3_{j1}$, so we may denote the corresponding parse tree by T_{ij}. In particular, all monomials in $\mathcal{J}(G)$ consist of exactly three variables when G is a 3-homogeneous st-DAG, and so each parse tree T_{ij} will contain exactly two product gates λ_{ij} and υ_{ij} such that υ_{ij} is an ancestor of λ_{ij}.

The three types (I, II, and III) of parse tree for monomials consisting of three indeterminates are shown in Figure 4. Note that every product gate is either υ_{ij} or λ_{ij} for some $i \in \{1, \ldots, n_1\}, j \in \{1, \ldots, n_2\}$.

We establish a canonical form for Φ in the following lemma.

Lemma 5.3. *For every accumulation circuit Φ for G, there exists an accumulation circuit Φ' for G such that $|\Phi'| = |\Phi|$ and $x^2_{ij} \in X_{\lambda_{ij}}$ for all $T_{ij} \in \mathrm{PT}(\Phi')$.*

Proof. We give an explicit construction for constructing the desired circuit Φ' from an arbitrary accumulation circuit Φ.

Let T_{ij} be any parse tree in $\mathrm{PT}(\Phi)$ such that $x^2_{ij} \notin X_{\lambda_{ij}}$. Let α, β be the children of υ_{ij} such that λ_{ij} is a descendant of α and x^2_{ij} is a descendant of β. Now observe that there must exist $L \in \mathrm{mon}(\lambda_{ij})$ such that $x^1_{1i}, x^3_{j1} \in L$. It follows that L is ϕ-unique, and Lemma 4.6 implies $|\mathrm{mon}(\beta)| = 1$, from which we may conclude that $\beta = x^2_{ij}$. As $\{x^2_{ij}\}$ is ϕ-unique, we may apply similar reasoning to conclude that $\alpha = \lambda_{ij}$ and $\mathrm{mon}(\lambda_{ij}) = \{x^1_{1i}, x^3_{j1}\}$. We now have that x^1_i, x^3_{j1} are the children of λ_{ij} and λ_{ij}, x^2_{ij} are the children of υ_{ij}. Since λ_{ij} is ϕ-unique, υ_{ij} is the sole parent of λ_{ij}. It follows that we may transform Φ in the following way without affecting the polynomial that it computes: (1) create a new product gate $\lambda_{ij}{}'$ with children x^1_{1i}, x^2_{ij};

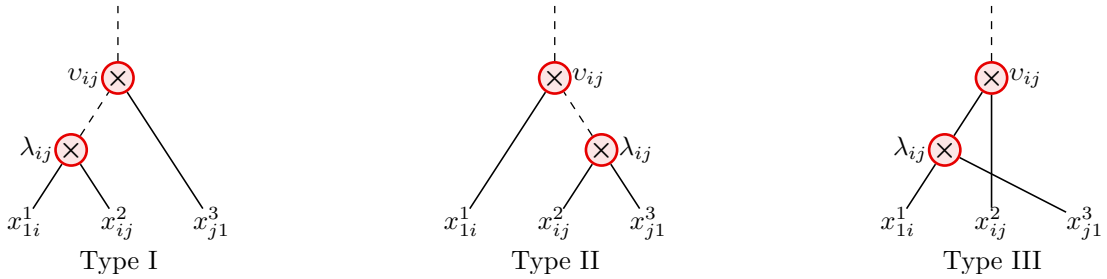


Figure 4: The parse tree $T_{ij} \in \mathrm{PT}(\Phi)$ corresponding to monomial $x^1_{1i}x^2_{ij}x^3_{j1} \in \mathrm{mon}(\phi)$ must be one of the three types shown here. A dashed edge indicates that one or more sum gates may be present on the path between the nodes indicated. In Lemma 5.3, we show that there is always an accumulation circuit of minimum size such that no parse tree is of type III.

(2) create a new product gate $\upsilon_{ij}{}'$ with children $\lambda_{ij}{}', x^3_{j1}$; (3) connect $\upsilon_{ij}{}'$ to the unique parent of υ_{ij}; and (4) remove λ_{ij} and υ_{ij} from Φ.

We may continue with this process to obtain Φ' as desired. Since, at each step, the number of product gates (and, in fact, the total number of gates) does not change, we have $|\Phi'| = |\Phi|$. This completes the proof. □

Theorem 4. *Commutativity has no power in this setting.*

Proof. Follows from the condition that Lemma 5.3 places on the gates λ_{ij}. □

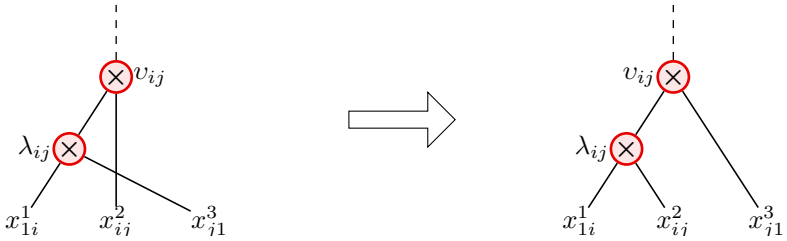


Figure 5: Every parse tree of type III (as on the left) can be transformed in this fashion without affecting the polynomial computed by the circuit.

A *vertex cover* for an undirected graph G is a set $C \subseteq V(G)$ of vertices such that every edge in $E(G)$ has at least one endpoint in C. The *vertex cover number* $\tau(G)$ of G is the smallest integer k for which there exists a vertex cover of size k for G.

Lemma 5.4. *If G is a 3-homogeneous st-DAG, then* $\mathbf{ACC}_\times(G) \geq \tau\big(G^2\big)$.

Proof. By Lemma 5.3, we can assume without loss of generality that Φ is an accumulation circuit for G in which $x^2_{ij} \in X_{\lambda_{ij}}$ for all $T_{ij} \in \mathrm{PT}(\Phi)$. Observe that the product gates in Φ can be partitioned into Λ and Υ, where $\Lambda = \{\lambda_{ij} \mid T_{ij} \in \mathrm{PT}(\Phi)\}$ and $\Upsilon = \{\upsilon_{ij} \mid T_{ij} \in \mathrm{PT}(\Phi)\}$ (note that Λ and Υ are disjoint). Since each $\lambda_{ij} \in \Lambda$ has x^2_{ij} as a child, we have $|\Lambda| = |X^2|$, from which $|\Upsilon| = k$ follows.

We now show how to use Υ to construct a vertex cover of size k for G^2. Since every parse tree T_{ij} is of type I or II, every $\upsilon_{ij} \in \Upsilon$ has either x^1_{1i} or x^3_{j1} as a child. We can thus construct the desired vertex cover of G^2 by iterating over the parse trees $T_{ij} \in \mathrm{PT}(\Phi)$, adding v^1_i if x^1_{1i} is a child of υ_{ij}, adding v^2_j otherwise. The constructed set of vertices constitutes a vertex cover, since for every edge $\{v^1_i, v^2_j\}$ in G^2 either v^1_i or v^2_j is added. Furthermore, since Υ consists of exactly k product gates, the constructed vertex cover will be of size k. This completes the proof of the lemma. □

6 The Upper Bound

Lemma 6.1. *If G is a 3-homogeneous st-DAG, then* $\mathbf{ACC}_\times(G) \leq |X^2| + \tau\big(G^2\big)$.

Proof. Let $H \subseteq V^1 \cup V^2$ be any vertex cover of G^2 that is *minimal*, meaning that no proper subset of H is also vertex cover. Our proof proceeds by induction on $|H|$.

Basis ($|H| = 1$). Suppose $|H| = 1$, and assume without loss of generality that $H = \{v_1^1\}$. It follows that $V^1 = \{v_1^1\}$ and, therefore,

$$\begin{aligned}\mathcal{J}(G) &= x_{11}^1 x_{11}^2 x_{11}^3 + x_{11}^1 x_{12}^2 x_{21}^3 + \cdots + x_{11}^1 x_{1n_2}^2 x_{n_2 1}^3 \\ &= x_{11}^1 \left(x_{11}^2 x_{11}^3 + x_{12}^2 x_{21}^3 + \cdots + x_{1n_2}^2 x_{n_2 1}^3\right) \ .\end{aligned}$$

We construct an accumulation circuit Φ with output ϕ in the natural way:

(i) Create product gates $\lambda_{1j} = x_{1j}^2 \times x_{j1}^3$ for $j = 1, \ldots, n_2$.

(ii) Create $|X^2| - 1$ sum gates to compute $\lambda_1^+ = \sum_j \lambda_{1j}$.

(iii) Create product gate $\phi = x_{11}^1 \times \lambda_1^+$.

We then have that ϕ computes $\mathcal{J}(G)$, and $|\Phi|_\times = |X^2| + 1$, as desired.

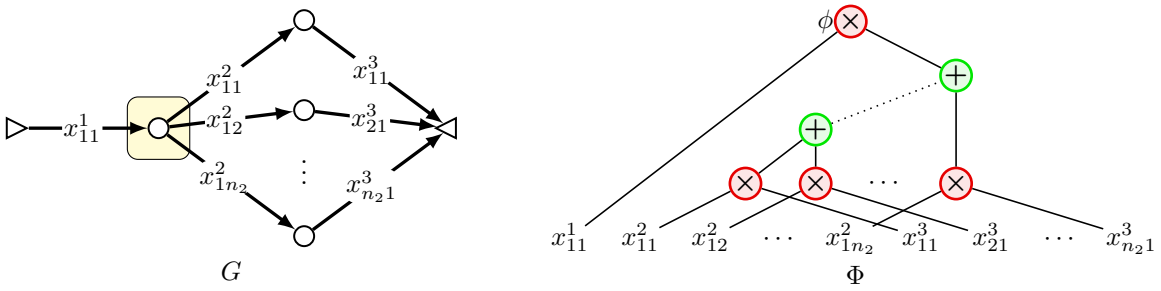


Figure 6: Basis step in the proof of Lemma 6.1.

Inductive step ($|H| \geq 2$). Now suppose $|H \geq 2|$, and assume without loss of generality that H contains the vertex v_1^1 and that the successors of v_1^1 in G are $v_1^2, \ldots, v_p^2$ where $p \leq |X^2|$. Define a new DAG $\tilde{G}$ by removing $\{v_i^1\}$ from G along with all incident arcs, where we can assume the resulting $\tilde{X}^1 \equiv X^1 \setminus \{x_{11}^1\} \neq \emptyset$ and $\tilde{X}^2 \equiv X^2 \setminus \{x_{11}^2, \ldots, x_{1n_2}^2\}$. Any $v_j^2 \in V^2$ left without any inarcs is also removed (along with incident arcs), so that $\tilde{G}$ is a 3-homogeneous *st*-DAG. Since no such v_j^2 could have been part of a minimal vertex cover, we have that $\tilde{H} \equiv H \setminus \{v_i^1\}$ is a vertex cover of $\tilde{G}^2$. Suppose (inductive hypothesis) $\mathbf{ACC}_\times(\tilde{G}) \leq |\tilde{X}^2| + |\tilde{H}|$, and let $\tilde{\Phi}$ be an accumulation circuit whose output $\tilde{\phi}$ computes $\mathcal{J}(\tilde{G})$ such that $|\tilde{\Phi}|_\times = |\tilde{X}^2| + |\tilde{H}| = |X^2| - p + |H| - 1$. Note that $\mathcal{J}(\tilde{G})$ is exactly $\mathcal{J}(G)$ without any of the monomials containing x_{11}^1. By applying the same construction as in the basis step, we can construct an accumulation circuit $\hat{\Phi}$ with output $\hat{\phi}$ that computes the missing monomials

$$\mathcal{J}(\hat{G}) = x_{11}^1 x_{11}^2 x_{11}^3 + x_{11}^1 x_{12}^2 x_{21}^3 + \cdots + x_{11}^1 x_{1p}^2 x_{p1}^3$$

such that $|\hat{\Phi}|_\times = p + 1$. Now create a new sum gate $\phi = \hat{\phi} + \tilde{\phi}$ to act as the output of an accumulation circuit Φ. Observe that Φ computes $\mathcal{J}(G)$ and $|\Phi|_\times = |X^2| + |H|$, implying that $\mathbf{ACC}_\times(G) \leq |X^2| + |H|$.

Since any vertex cover H of minimum size must also be minimal, we have that $\mathbf{ACC}_\times(G) \leq |X^2| + \tau\big(G^2\big)$, as desired. □

Theorem 5. *The following hold for every 3-homogeneous st-DAG G.*

(i) $\mathbf{ACC}_\times(G) = |X^2| + \tau\big(G^2\big)$.

(ii) $\mathbf{ACC}_+(G) = |X^2| - 1$.

(iii) $\mathbf{ACC}(G) = 2|X^2| + \tau\big(G^2\big) - 1$.

Proof. Follows from Lemma 5.2, Lemma 5.4, and Lemma 6.1. □

7 The Optimal Structural Derivative Accumulation Problem

Theorem 6. *There exists a polynomial-time algorithm that, given a DAG G, constructs an accumulation circuit Φ computing $\mathcal{J}(G)$ such that $|\Phi|_\times = \mathbf{ACC}_\times(\mathcal{J}(G))$ and $|\Phi_+| = \mathbf{ACC}_+(\mathcal{J}(G))$.*

Proof. The proof for the upper bound presented in Lemma 6.1 describes a constructive algorithm for obtaining Φ from a vertex cover H. The described procedure can be made to run in polynomial time; choose an arbitrary cover vertex at each step. Since an optimal vertex cover of a bipartite graph can also be found in polynomial time, it follows that we can construct the desired accumulation circuit Φ in the same time bound. □

In general, we may be given any DAG G as input. We are especially interested, then, in whether there is an algorithm that, given a DAG G, can produce an accumulation circuit that evaluates $\mathcal{J}(G)$ by using only $\mathbf{ACC}_\times(\mathcal{J}(G))$ multiplications. The corresponding decision problem, formalized below as a circuit minimization problem, has been conjectured for decades to be NP-hard.

Optimal Structural Derivative Accumulation
Instance: DAG G, integer K.
***Question*:** Is $\mathbf{ACC}_\times(\mathcal{J}(G)) \leq K$?

A number of heuristics have been proposed and implemented for this problem [6, 16, 17]; many work by successively performing local transformations to G ("eliminations"), eventually reducing G to a (directed) bipartite graph with arcs corresponding to the nonzero entries of $\mathcal{J}(G)$. An important property that these techniques share is that they do not exploit the commutativity of the underlying field $\mathbb{R}$.

Naumann [18] considers a variant of Optimal Structural Derivative Accumulation in which two or more arcs in G may be labeled with the same indeterminate. Note that, when this is permitted, *any* collection of polynomials can be expressed as $\mathcal{J}(G)$ for some DAG G. This is accomplished by creating a collection of edge-disjoint paths, one for each monomial. Naumann is thus able to show that this variant is NP-hard[1] via a reduction from ensemble computation [2]. The Optimal Structural Derivative Accumulation problem concerns the complexity that arises from the *structure* of the chain rule (as manifest in G). In the remainder of this section, we consider a different type of generalization—one that allows us to relate the complexity of $\mathcal{J}(G)$ to that of another function representing a multilinear polynomial: bilinear forms.

7.1 Unit arcs and bilinear forms

In this section, we show that the following variant of Optimal Structural Derivative Accumulation is NP-hard. Instead of labeling *every* arc from $A(G)$ with its own indeterminate from X, we label *some* of the arcs with indeterminates and the rest with the multiplicative identity 1. We still assume that there are no algebraic dependences among the indeterminates that label edges, so the structure of G is somewhat preserved in $\mathcal{J}(G)$. Note that *any* DAG can be made k-homogeneous for some k polynomial in $|A(G)|$ when unit arcs are permitted. Griewank and Naumann [5] implicitly use this fact in formulating a heuristic for the Optimal Structural Derivative Accumulation problem, where $\mathcal{J}(G)$ is represented as a chained product of sparse matrices with entries from $X \cup \{0, 1\}$, and a variant of the dynamic programming approach to chained matrix bracketing is applied. However, we note that when a k-homogeneous DAG G contains unit arcs, the resulting collection of polynomials is no longer necessarily k-homogeneous. Indeed, in the next section we discuss 3-homogeneous DAGs G for which $\mathcal{J}(G)$ happens to be a 2-homogeneous polynomial.

[1] Though Naumann asserts that Optimal Structural Derivative Accumulation is in NP, this has not been shown. Observe that the number of paths in a DAG can be exponential in the number of arcs, in which case there will be an exponential number of terms in the polynomials composing $\mathcal{J}(G)$. For this problem to be in NP, we must be able to check in polynomial time whether an accumulation circuit Φ computes $\mathcal{J}(G)$ for a given DAG G; it is not immediately clear that this is possible.

Bilinear forms. Let $\mathbf{a}$ and $\mathbf{b}$ be vectors of indeterminates of dimension p and q, respectively, and let $M \in \mathbb{F}^{p\times q}$ be a matrix of field elements. A *bilinear form* f over $\mathbf{a}$ and $\mathbf{b}$ is defined as the product $f = \mathbf{a}^\top M\mathbf{b}$. Gonzalez and JáJá [3] studied the complexity of monic bilinear forms (those for which $M \in \{0,1\}^{p\times q}$). In fact, they considered only computations in the polynomial semiring $\{0,1\}[X]$, which corresponds exactly to our model of accumulation circuits. The *biclique cover number* of an undirected graph G is the smallest number k of complete bipartite graphs into which the edge set $E(G)$ can be partitioned. The following result was established by a reduction from the biclique cover problem on bipartite graphs, which is shown to be NP-complete earlier in the same paper.

Theorem 7 ([3])**.** *Given a bilinear form f over $\{0,1\}[X]$ and a positive integer K, the problem of determining whether $\mathbf{ACC}_\times(f) \le K$ is* NP*-complete.*

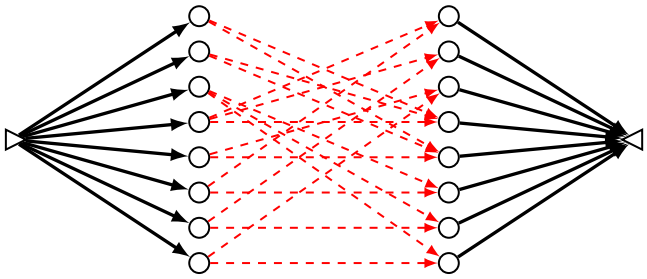
Given a bilinear form f, a DAG G for which $\mathcal{J}(G) = f$ can be constructed in the manner laid out in Section 3. Such a DAG will be a 3-homogeneous *st*-DAG, where X^2 is a fixed 0-1 matrix rather than consisting of indeterminates.

Corollary 8. *The variant of* Optimal Structural Derivative Accumulation *in which a subset of the arcs may be labeled with the multiplicative unit 1 is* NP*-hard and remains so even when G is a 3-homogeneous st-DAG.*

Note that adding unit arcs can also change dramatically the types of operations that an accumulation procedure for $\mathcal{J}(G)$ may consist of. For example, Figure 7 demonstrates the utility of operations such as

$$\left(x^1_{11} + x^1_{12}\right)\left(x^3_{41} + x^3_{51}\right),$$

which are not useful when $\mathcal{J}(G)$ is considered for a DAG G without unit arcs.



G

$$X^2 = \begin{bmatrix} 0&0&0&1&1&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&0&1&1&1\\ 1&1&0&1&0&0&0&0\\ 0&0&1&0&1&0&0&0\\ 1&0&0&0&0&1&0&0\\ 0&1&0&0&0&0&1&0\\ 0&0&1&0&0&0&0&1 \end{bmatrix}$$

$$\begin{aligned}
\mathcal{J}(G) &= x^1_{11}x^3_{41} + x^1_{11}x^3_{51} + x^1_{12}x^3_{41} + x^1_{12}x^3_{51} + x^1_{13}x^3_{61} + x^1_{13}x^3_{71} + x^1_{13}x^3_{81} + x^1_{14}x^3_{11} + x^1_{14}x^3_{21}\\
&\quad + x^1_{14}x^3_{41} + x^1_{15}x^3_{31} + x^1_{15}x^3_{51} + x^1_{16}x^3_{11} + x^1_{16}x^3_{61} + x^1_{17}x^3_{21} + x^1_{17}x^3_{71} + x^1_{18}x^3_{31} + x^1_{18}x^3_{81}\\
&= \left(x^1_{11} + x^1_{12}\right)\left(x^3_{41} + x^3_{51}\right) + x^1_{13}\left(x^3_{61} + x^3_{71} + x^3_{81}\right) + x^1_{14}\left(x^3_{11} + x^3_{21} + x^3_{41}\right)\\
&\quad + x^1_{15}\left(x^3_{31} + x^3_{51}\right) + x^1_{16}\left(x^3_{11} + x^3_{61}\right) + x^1_{17}\left(x^3_{21} + x^3_{71}\right) + x^1_{18}\left(x^3_{31} + x^3_{81}\right)\\
&= \left(x^1_{11} + x^1_{12} + x^1_{14}\right)\left(x^3_{11} + x^3_{21} + x^3_{41}\right) + \left(x^1_{13} + x^1_{16}\right)\left(x^3_{11} + x^3_{61}\right)\\
&\quad + \left(x^1_{11} + x^1_{12} + x^1_{15}\right)\left(x^3_{31} + x^3_{51}\right) + \left(x^1_{13} + x^1_{17}\right)\left(x^3_{21} + x^3_{71}\right)\\
&\quad + \left(x^1_{13} + x^1_{18}\right)\left(x^3_{23} + x^3_{81}\right) - \left(x^1_{11} + x^1_{12} + x^1_{13}\right)\left(x^3_{11} + x^3_{21} + x^3_{31}\right)
\end{aligned}$$

Figure 7: An example due to Gonzalez and JáJá [3]. Seven multiplications are necessary and sufficient to evaluate $B = X^1X^2X^3$ over $\{0,1\}[X]$. This is also an example of an *st*-DAG for which $\mathbf{C}_\times(\mathcal{J}(G)) < \mathbf{ACC}_\times(\mathcal{J}(G))$ (shown using six multiplications over $\{-1,0,1\}[X]$).

Problem	Function	$\otimes$	$\oplus$	idempotent?
Derivative Accumulation	$\mathbb{R}^{\lvert E\rvert} \to \mathbb{R}$	$*$	$+$	
$\#s$-t Paths	$\{0,1\}^{\lvert E\rvert} \to \mathbb{N}$	$*$	$+$	$\times$
Shortest s-t Path	$\mathbb{R}^{\lvert E\rvert} \to \mathbb{R}$	$+$	min	$+$
s-t Connectivity	$\{0,1\}^{\lvert E\rvert} \to \{0,1\}$	$\wedge$	$\vee$	$\times, +$

8 Discussion

8.1 The power of commutativity

We saw that exploiting associativity and distributivity were crucial in achieving the tight bound on the complexity of $\mathcal{J}(G)$ for 3-homogeneous st-DAGs. At the same time, Theorem 4 shows that, for this class of graphs, the complexity of $\mathcal{J}(G)$ does not depend on whether the indeterminates commute. Likewise, commutativity is not used in the monotone circuits for matrix multiplication discussed in Theorem 1 and Theorem 2, where the schoolbook method is used.

Naumann and Griewank conjecture the following.

Conjecture 1 ([4, 17])**.** *For every DAG G, there exists an optimal accumulation circuit computing $\mathcal{J}(G)$ that does not use commutativity.*

In fact, they conjecture something a bit stronger: that the elimination technique known as *face elimination* can always be used to obtain an optimal accumulation circuit. Griewank [4] shows that Conjecture 1 is true for DAGs that are *absorption-free*, meaning that there is at most one directed path between any two vertices. Accumulation circuits for such DAGs consist of product gates exclusively.

8.2 Other semirings

Disallowing commutativity can be viewed as a restriction of the algebraic properties of the semiring $\langle \mathbb{R}, \times, +, 0, 1\rangle$ that we consider. At the opposite end of the spectrum lies the Boolean semiring $\langle \mathbb{Z}_2, \wedge, \vee, 0, 1\rangle$, where $\mathcal{J}(G)$ computes the all-source all-sink connectivity function. This semiring is much more general, in the sense that any monotone arithmetic circuit for $\mathcal{J}(G)$ will immediately become a monotone Boolean circuit for many-source, many-sink connectivity when the operation $\times$ is replaced with $\wedge$ and $+$ with $\vee$. It is striking that, as Theorem 2 shows, the complexity of dense $n \times n$ is exactly the same for such a wide range of semirings. In general, lower bounds for connectivity translate directly into lower bounds for any ring that is more restrictive. Likewise, upper bounds on structural derivative accumulation apply to less restrictive semirings. Iri [9] shows that forward and reverse modes are equivalent to an algorithm in $\langle \mathbb{R}, \min, +, \rangle$, an only slightly more general semiring than our own. In this context, $\mathcal{J}(G)$ computes many-source, many-sink shortest paths. This connection is also discussed by Mahr [12, 13], and is considered by Mehlhorn [14] for general directed graphs (those not necessarily acyclic). In this sense, our algorithms can be viewed as a preprocessing step for answering a large number of connectivity or shortest path queries, and so forth.

Definition 12 (idempotence)**.** *An operation $\circ$ is* idempotent *if $x \circ x = x$ for all $x \in \mathbb{F}$.*

Examples of idempotent operations include min and max over the field $\mathbb{R}$ and $\wedge$ and $\vee$ over the field $\mathbb{Z}_2$.

Observation 8.1. *Let Φ be an accumulation circuit over a semiring $\langle \mathbb{F}, \otimes, \oplus, 0, 1\rangle$. If $\otimes$ is idempotent, then Φ is multilinear.*

Observation 8.2. *Let Φ be a multilinear accumulation circuit over a semiring $\langle \mathbb{F}, \otimes, \oplus, 0, 1\rangle$. If $\oplus$ is idempotent, then Φ is monic.*

An example of a semiring over $\mathbb{R}$ for which both $\otimes$ and $\oplus$ are idempotent is $\langle \mathbb{F}, \min, \max, -\infty, +\infty\rangle$. See Table 8.2 for more examples.

For monotone computations over indeterminates that take 0-1 values [23] the $\otimes$ operation is idempotent. In this case, $\mathcal{J}(G)$ counts the number of paths.

8.3 The power of cancellation

This is the only part of the paper where we consider cancellations. We make note here of two small concrete examples for which results are known that illustrate the utility of cancellations in evaluating $\mathcal{J}(G)$. Strassen's algorithm [24] for multiplying 2×2 matrices (see Figure 1) uses only seven multiplications, while eight are required in the monotone model. (Note that an essential part of this algorithm is that it is noncommutative.) Gonzalez and JáJá [3] give an example of an 8×8 bilinear form (Figure 7) that requires seven multiplications over $\{0, 1\}[X]$ and can be done with six over $\{-1, 0, 1\}[X]$.

Acknowledgments

I am indebted to Andreas Griewank, Uwe Naumann, Alexander Razborov, and Jean Utke for numerous discussions related to the material presented here. This work was supported in part by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] F. Bauer. Computational graphs and rounding error. *SIAM Journal on Numerical Analysis* 11(1):87–96, 1974, doi:10.1137/0711010. 1

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W.H. Freeman and Company, San Francisco, 1979. 10

[3] T. F. Gonzalez and J. JáJá. On the complexity of computing bilinear forms with $\{0, 1\}$ constants. *Journal of Computer and System Sciences* 20(1):77–95, 1980, doi:10.1016/0022-0000(80)90006-9. 11, 13

[4] A. Griewank. A mathematical view of automatic differentiation. *Acta Numerica*, vol. 12, pp. 321–398. Cambridge University Press, 2003, doi:10.1017/S0962492902000132. 12

[5] A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Mathematical Programming* 3(95):555–571, 2003, doi:10.1007/s10107-002-0329-7. 10

[6] A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markowitz rule. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pp. 126–135. SIAM, 1991. 2, 10

[7] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Other Titles in Applied Mathematics 105. SIAM, Philadelphia, 2nd edition, 2008. 1

[8] L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical Report 0300, INRIA, 2004, `http://www.inria.fr/rrrt/rt-0300.html`. 2

[9] M. Iri. Simultaneous computation of functions, partial derivatives, and estimates of rounding errors —complexity and practicality—. *Japan Journal of Applied Mathematics* 1(2):223–252, 1984, doi:10.1007/BF03167059. 12

[10] M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM* 29(3):874–897, 1982, doi:10.1145/322326.322341. 4, 7

[11] M. Mahajan and B. V. R. Rao. Small-space analogues of Valiant's classes. *FCT*, pp. 250–261. Springer, Lecture Notes in Computer Science 5699, 2009, doi:10.1007/978-3-642-03409-1_23. 5

[12] B. Mahr. A birds eye view to path problems. *WG*, pp. 335–353. Springer, Lecture Notes in Computer Science 100, 1980, doi:10.1007/3-540-10291-4_25. 12

[13] B. Mahr. Algebraic complexity of path problems. *Informatique Théorique* 16(3):263–292, 1982. 3, 12

[14] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Monographs in Theoretical Computer Science, an EATCS Series 2. Springer, 1984, http://www.mpi-sb.mpg.de/~mehlhorn/DatAlgbooks.html. 4, 12

[15] K. Mehlhorn and Z. Galil. Monotone switching circuits and Boolean matrix product. *Computing* 16(1-2):99–111, 1976, doi:10.1007/BF02241983. 4

[16] U. Naumann. *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs*. Ph.D. thesis, Technical University of Dresden, December 1999. 10

[17] U. Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Mathematical Programming, Ser. A* 99(3):399–421, 2004, doi:10.1007/s10107-003-0456-9. 10, 12

[18] U. Naumann. Optimal Jacobian accumulation is NP-complete. *Mathematical Programming* 112(2):427–441, 2008, doi:10.1007/s10107-006-0042-z. 10

[19] N. Nisan and A. Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity* 6(3):217–234, 1997, doi:10.1007/BF01294256. 5

[20] M. S. Paterson. Complexity of monotone networks for Boolean matrix product. *Theoretical Computer Science* 1(1):13–20, 1975, doi:10.1016/0304-3975(75)90009-2. 4

[21] V. R. Pratt. The power of negative thinking in multiplying Boolean matrices. *STOC '74: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, pp. 80–83. ACM, 1974, doi:10.1145/800119.803887. 4

[22] R. Raz and A. Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity* 18(2):171–207, 2009, doi:10.1007/s00037-009-0270-8. 5

[23] R. Sengupta and H. Venkateswaran. A lower bound for monotone arithmetic circuits computing 0-1 permanent. *Theoretical Computer Science* 209(1-2):389–398, 1998, doi:10.1016/S0304-3975(98)00130-3. 12

[24] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik* 13(4):354–356, 1969, doi:10.1007/BF02165411. 13

[25] J. Utke, U. Naumann, and A. Lyons. OpenAD/F: User Manual. Tech. rep., Argonne National Laboratory, http://www.mcs.anl.gov/OpenAD/openad.pdf. 2

[26] V. Vassilevska. *Efficient Algorithms for Path Problems in Weighted Graphs*. Ph.D. thesis, Carnegie Mellon University, August 2008, http://reports-archive.adm.cs.cmu.edu/anon/2008/CMU-CS-08-147.pdf. 3

[27] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms* 1(1):2–13, 2005, doi:10.1145/1077464.1077466. 3

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.